Jam Tomorrow: Collaborative Music Generation in Croquet Using OpenAL

Florian Thalmann
thalmann@students.unibe.ch

Markus Gaelli
gaelli@iam.unibe.ch
Institute of Computer Science and Applied Mathematics, University of Bern
Neubrückstrasse 10, CH-3012 Bern, Switzerland

Abstract

[1] *We propose a music generation software that allows large numbers of users to collaborate. In a virtual world, groups of users generate music simultaneously at different places in a room. This can be realized using OpenAL sound sources. The generated musical pieces have to be modifiable while they are playing and all collaborating users should immediately see and hear the results of such modifications. We are testing these concepts within Croquet by implementing a software called Jam Tomorrow.*

1. Introduction

"It's very good jam" said the Queen.
"Well, I don't want any *today*, at any rate."
"You couldn't have it if you *did* want it," the Queen said. "The rule is jam tomorrow and jam yesterday but never jam *today*."
"It *must* come sometimes to 'jam today,"' Alice objected.
"No it can't," said the Queen. "It's jam every *other* day; today isn't any *other* day, you know."
"I don't understand you," said Alice. "It's dreadfully confusing."
Lewis Carroll, Through the Looking Glass, 1871.

Even though the idea of *collaborative music generation on computer networks* is almost thirty years old, today there are only a few projects in this domain [15]. Improvisation over the web has always been technically constrained, due to traffic limitations and synchronicity problems, but is now possible using MIDI [3] and even using compressed audio [14]. There are also projects about collaborative composing [15] [13] [1] or collaborative open source recording [12]. People using such compositional or improvisational software are restricted to working on one musical piece at a time. Our idea is to develop a software that lets users collaborate in generating multiple musical pieces simultaneously within a modern virtual world. People using this software will have compositional and improvisational possibilities.

In this paper, we start by presenting our conceptual ideas. In section 3 we explain the technology we use for our explorative project, Jam Tomorrow. We continue in section 4 by giving you a more detailed description of the musical concept and the architecture of Jam Tomorrow. In the same section we describe the major problems we encountered during implementation. Finally, we conclude by discussing future work.

2. Concept: Multiple Musicians in One Virtual Room

Our idea is to build an application that consists of several similar *musical editors* (graphical user interfaces) distributed within a virtual room. Each of these editors is linked to a musical piece and can be used as an interface for modifying it. The corresponding musical piece is heard from a sound source located at the editor's position.

The virtual room is visited by multiple users, which can move around freely between the editors. Depending on a user's distance from an editor, volume and panning of this editor's music changes. So, a user can find the editor of the music he is interested in, simply by following sound sources.

Every user can use any editor. If somebody modifies a musical piece, this has to be immediately audible to all users near its editor. Of course, this constrains the musical possibilities a lot. For our ideas of such a flexible and collaborative music, see section 4.2.

[1] Fourth International Conference on Creating, Connecting and Collaborating through Computing (C5'06) pages 73–78

With our concept, we can have a *large number of tunes in one huge musical world*. Users in this world can collaborate in groups or work alone, modify an existing tune or start a new one. It is certainly very interesting to walk through this world and hear the editors play the music reflecting the common likes of the users.

3. Technology: Croquet Project and OpenAL

OpenAL is an API that "allows a programmer to position audio sources in a three-dimensional space around a listener, producing reasonable fading and panning for each source so that the environment seems three-dimensional" [6]. This is exactly what we need for the placement of our musical pieces within the room. Unfortunately, at the time, OpenAL does not support MIDI yet, so we are restricted to *sampled sound* [2].

"*Croquet* is a combination of open source computer software and network architecture that supports deep collaboration and resource sharing among large numbers of users" [16] and it is implemented in Squeak. Although Croquet is still being developed, it seems to be the perfect framework for the integration of our application.

A Croquet environment is divided into spaces, in which users can move around using their avatar. Unlike in client-server applications, each participant in a Croquet network session, has an independent image holding the entire application on his computer. At the moment, for establishing a connection between Croquet users, all images have to be in the same state and contain the same program code. To keep the images synchronized, every message sent to a Croquet object in a space, is replicated in the other participants image. To replicate a message manually, the #meta message has to be sent. For instance, 'self update' becomes 'self meta update'. For understanding our architecture, it is important to know these principles.

3.1. Croquet's OpenAL Interface and its Performance

An implementation of OpenAL is already integrated in Croquet and the sampled sound file formats *wav*, *aif* and *mp3* are supported. During the development of Jam Tomorrow, we explored the possibilities of Croquet's OpenAL interface. Before talking about the implementation of our project, we would like to discuss the performance limitations we have.

The basic OpenAL Croquet class we use is *TSound*. A TSound holds a SoundBuffer containing a sampled sound and manages the OpenAL buffers and sources needed for playing. It also understands common control messages like #play, #pause and #stop. One can assign a TFrame (a Croquet object with a position) to a TSound, so that when this TSound is played, its OpenAL source is located at the TFrame's position in the Croquet space.

TSound or similar classes like OpenALStreamingSound, are currently the only way to use OpenAL in Croquet. If we want to play a sound, we have to copy its sound buffer to a TSound object's sound buffer and then start streaming the TSound (fill the OpenAL buffers). We have to do this once, if we plan to use the same sound repeatedly. But if we want to play different sounds, we have to copy the buffer and stream the sound for every new sound.

If we do this for short sounds a user has saved or generates locally, performance is fairly good. But performance gets worse, as soon as sounds have to be sent over the network. For example with the current voice chatting implementation (TVoiceRecorder), when somebody records audio, it takes too much time for the sound to reach the other participant's computers. For musicians playing simultaneously, latency time needs to be very close to zero.

Due to these problems, we are limited to using locally saved or generated sounds. Improvisation over network using microphones is therefore not yet possible. To integrate realtime MIDI functions using TSounds would result in an even more unsatisfying performance. We have experimented with Squeak's FMSynth, but the translation to TSounds is too slow for complex MIDI messages. However, it could be possible to build a simple MIDI application, where every note is pregenerated as TSound.

We hope, that OpenAL will soon support MIDI for this would increase our musical flexibility dramatically. There are software products that translate MIDI to audio [17], but not yet in realtime.

At the moment, Croquet cannot be used for virtual band playing in realtime yet. This is why we chose to build a first application that does not need to be strongly synchronous. In our concept it is important, that every participant hears exactly the same music, but not necessarily at the same time. For example, it is possible, that the user who presses the 'play'-button, hears the music a bit earlier than his fellow users.

4. Implementation: Jam Tomorrow

For experimentation with Croquet and OpenAL, we have built a simple prototype called Jam Tomorrow. It consists of a number of editors (GUI windows), each of them linked to a sound player. Users can add different types of tracks to a player, modify and play them. We also integrated a function to record samples and add them to the player. But this function cannot be used yet, due to its poor performance.

4.1. A Sample Scenario

As an example, we have a Croquet space with three editors and four user avatars in it (see Fig. 1). Jane and John collaborate modifying the musical piece in editor 1. We will see later, what exactly they are doing. They both hear the music they are producing in editor 1, but they also hear Max's music in editor 2 (Jane a bit louder than John). Max prefers to make his own music, but if somebody else wants to join him, he has to be happy with it. Min enjoys the situation doing nothing and listening to the different changing musical pieces. The music in editor 3 is not playing at the time, so she hears editor 1 on her right hand side and editor 2 on her left hand side.

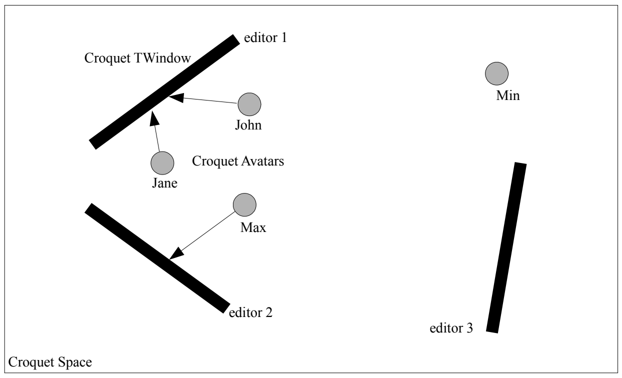


Figure 1. A space with avatars and editors.

The important thing now, is how Jane and John are able to collaborate: Jane decides to make the music sound more interesting. She selects a sound file that fits the musical context. So she adds this file using editor 1. She presses the generate button and the sound file is transformed into a loop sample (see next section). Meanwhile, John sees her pointer moving, so he is able to watch everything she is doing. When the sample is ready, John hears it, integrated in the music.

But John, does not like the way the sample sounds. He thinks it is too loud and too simple. So first he turns down the sample's volume and then decides to choose another algorithm to generate the sample. He prefers experimental music and therefore lets a random fragmentation algorithm randomize the sample. The first recalculation does not fit the music at all, so Jane chooses to press the generate button again. Now, they are lucky: the new sample is so beautiful that Min decides to join them. She has some incredible ideas that will change John and Jane's lives.

4.2. Musical Concept

Even though, for our explorative prototype, we developed a rather simple musical concept, this is just a suggestion that may be replaced or expanded in later development. However, this concept is simple to implement and fulfills the following needs of a flexible musical form.

Approaching one of the editors, a user should immediately hear what the music generated there is about. Then, when somebody modifies this music, the results should be heard immediately. Additionally, the music should be modifiable while it is playing. We thought of two musical forms guaranteeing such flexibility, while being simple enough.

- The first form has become very popular since the electrification of music: the *loop form*. It lets us generate "infinite" musical pieces that always continue playing (looping), even while they are being edited. So, all modifications can be audible immediately, because there are no long formal structures to wait for.
- The second form is total *improvisation*. Users participate live using a microphone (or a MIDI interface). However, with the current Croquet implementation, it is impossible to integrate an improvisational function for microphone or MIDI interfaces, due to synchronicity problems.

Jam Tomorrow is based on the loop form, although we extended it for experimental reasons. We also integrated the possibility to generate Musinum tracks. Musinum [4] [8] is an algorithmic music generation software based on fractals. See section 4.4 for information about the implementation of our musical concept.

4.3. The Jam Tomorrow Architecture

During development, we encountered several problems, mainly with the Croquet concept. Our goal was to use unmodified Croquet and Squeak parts, but we found out that what we were planning to implement was not possible without changing some methods in the Croquet framework.

What we have now, is an architecture that can be seen as an application of the MVC (Model-View-Controller) pattern [11] in Croquet. We have a model consisting of the player and its tracks. It is the responsibility of this model to keep the corresponding models of the other images in the network synchronized (see Fig. 2). Each model has one view by default, which is updated automatically. The views in the different images only know their own models, so they are separated from each other.

This architecture with separated views does not fit the Croquet concept perfectly. However, this is not noticeable for users, because the views stay synchronized through their

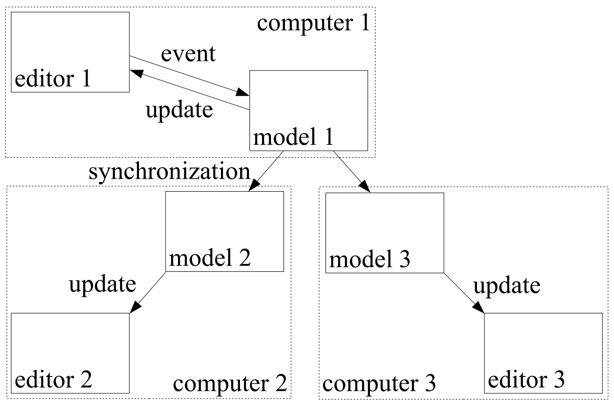


Figure 2. Jam Tomorrow's architecture in action.

models. Additionally, we have three advantages using such an architecture.

- Croquet classes managing their own synchronization, i.e. having methods containing meta sends (TSound, for example), can be used unchanged (see section 4.5).
- We are able to program nondeterministic functions, where the model, which receives the message from its view, first calculates the nondeterministic part (Fig. 3). Then, the first model updates the other models with the result, and each model continues calculation on its own. Finally, each view is updated by its model. This concept is used in FragmentedSample (see section 4.4.1).

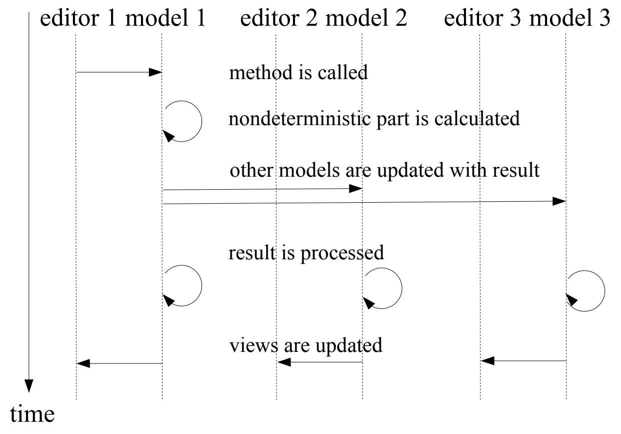


Figure 3. A sequence diagram for nondeterministic functions.

- Separated functions can be attached directly to the GUI (musical editor). A GUI button, for example, can have a function that only has a local effect. Usually in Croquet, for such functionality, an overlay button is added to the global Croquet user interface. However, our method keeps the musical editor's UI more compact and simple. We used this concept for the sample recording function. If a user presses the record button, recording starts on his computer only.

4.4. The Model Classes

Our actual version of Jam Tomorrow can be seen as an *"intelligent" loop device*. We have a *JamTomorrowPlayer*, which holds a collection of *JamTomorrowTracks*. There are two kinds of tracks: *LoopSamples* and *MusinumTracks*, both of which have a specified playing probability. Users can define how often a loop should be played.

LoopSamples are based on sampled sounds imported from a specified sound file. When imported, these sampled sounds are transformed to fit the actual loop length of the JamTomorrowPlayer. This length is defined using the two values bpm (beats per minute, reflects the tempo of the musical piece) and bpl (beats per loop, defines the length of a loop in beats). LoopSamples are played in a loop cycle. At the beginning of every cycle, all playing LoopSamples are stopped. The LoopPlayer then decides for each sample if it should be played or not, based on the sample's probability.

A JamTomorrowPlayer may hold a variety of different LoopSample types. We propose four kinds of samples, the trivial *SimpleSample* and *TiledSample*, and the more complicated *StretchedSample* (not yet implemented) and *FragmentedSample*. Each sample type needs different specific parameters to be defined by the users.

MusinumTracks have no unified length. They simply start to play when the player starts and continue playing until either they reach their end or the player stops (see [8] for more information about Musinum). Due to the incompatibility of MIDI and Croquet, we have integrated Musinum into Jam Tomorrow using the interface of Squeak's FMSynth. Each played note is generated live with the FMSynth. If we have many tracks, this implementation becomes rather slow, but it is the only way to integrate a software like Musinum.

4.4.1 Random Numbers in Croquet

For the fragmentation of FragmentedSamples, we use random numbers. This makes them sound very interesting. But the use of random numbers causes problems in Croquet. Usually, if we send a message to a random number generator, this message is replicated on each computer. We get a different number on each computer and the images are not synchronized anymore. This problem is discussed on site [10].

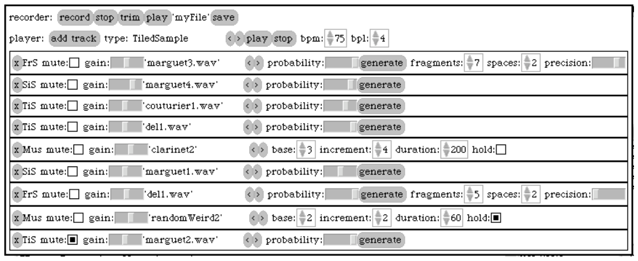
We have come up with another solution, made possible by Jam Tomorrow's architecture. The user interfaces on the different computers are separated from each other and update themselves following their models, as described above. Like this, we can let one model calculate a random number, which then updates the corresponding models on the other computers with the result (by using the #meta message). Then, the function processing the random number is called from the first computer (also with #meta).

4.4.2 Sound Cosmetics

Using procedures like the one in FragmentedSample, where sound files are cut in smaller segments and reorganized, a crackling noise appears during playback. A common technique to prevent such noise used by professional audio recording software like Logic [9], is *crossfading*. We implemented a simple fading algorithm applied to each fragment moved to the sound buffer to smoothen the generated music.

4.5. A Croquet GUI

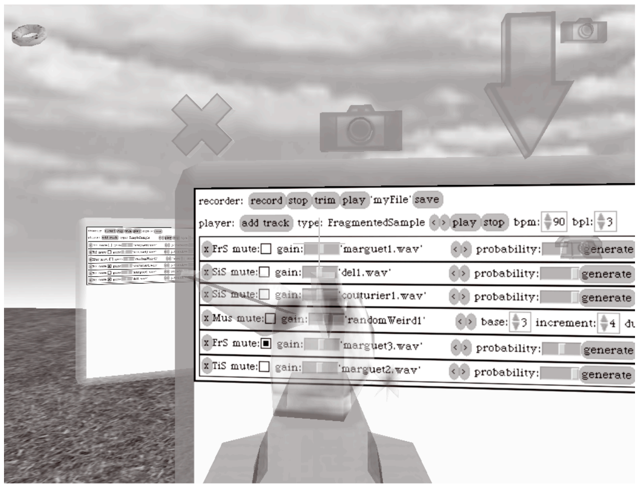
The actual GUI design is similar to many interfaces of music production softwares (see Fig. 4). There is an area with *control buttons*, where users can change global musical settings, add tracks and play or stop the music. Below these buttons, each added track is represented by a row. Such a row holds text fields, showing the settings of the concerned track, and different buttons, to change the track settings and to remove the track. For each track type, there is a different kind of row, providing specific controls.



```
recorder: record stop trim play 'myFile' save
player: add track  type: TiledSample         < > play stop  bpm: 75  bpl: 4
x FrS mute:☐ gain: 'marguet3.wav'    < > probability: generate fragments: 7 spaces: 2 precision:
x SiS mute:☐ gain: 'marguet4.wav'    < > probability: generate
x TiS mute:☐ gain: 'couturier1.wav'  < > probability: generate
x TiS mute:☐ gain: 'del1.wav'        < > probability: generate
x Mus mute:☐ gain: 'clarinet2'       < > base: 3 increment: 4 duration: 200 hold:☐
x SiS mute:☐ gain: 'marguet1.wav'    < > probability: generate
x FrS mute:☐ gain: 'del1.wav'        < > probability: generate fragments: 5 spaces: 2 precision:
x Mus mute:☐ gain: 'randomWeird2'    < > base: 2 increment: 2 duration: 60 hold:■
x TiS mute:■ gain: 'marguet2.wav'    < > probability: generate
```

Figure 4. The GUI Morph.

It is easy to integrate a Squeak project or morph into a Croquet space by using *TWindows* (see Fig. 5) in combination with TMorphics and TMorphMonitors. Any Squeak morph can be shown in such a window, the size, orientation and position of which users can change directly within the space. If we attach an OpenAL sound source to the window adn the window is moved, the source is automatically moved with it.



```
recorder: record stop trim play 'myFile' save
player: add track type: FragmentedSample < > play stop bpm: 90 bpl: 3
x FrS mute:☐ gain: 'marguet1.wav'     < > probability: generate
x SiS mute:☐ gain: 'del1.wav'         < > probability: generate
x SiS mute:☐ gain: 'couturier1.wav'   < > probability: generate
x Mus mute:☐ gain: 'randomWeird1'     < > base: 3 increment: 4
x FrS mute:■ gain: 'marguet3.wav'     < > probability: generate
x TiS mute:☐ gain: 'marguet2.wav'     < > probability: generate
```

Figure 5. An avatar in front of two GUI TWindows.

Usually, all events reaching this window on one machine are caught by its TMorphic and replicated on the other machines using #meta sends. To follow our architecture concept, we had to disable this automatic replication, so that the events just reach the local model. If we had not, it would have been impossible to use a TMorphic/TMorphMonitor-GUI in combination with the classes our model uses (for example TSound), as these classes are using the #meta message themselves.

The #addTrack method in JamTomorrowPlayer, for instance, uses meta for guaranteeing that the tracks, generated at the same time in the corresponding models of the network, are seen as corresponding tracks. When this method is linked to a button in a TWindow-GUI and the button is pressed, on each participating machine the #addTrack method containing the #meta send is executed. Because of the *double use of meta*, the final message (#addTrack: aTrack) is called one time per participant on each machine. So each participant gets multiple new tracks, instead of one.

In our architecture with the separated GUIs, when a user presses a GUI button, the #addTrack method is called just on his machine, and only then, controlled by the model using meta, the other participant's players get new tracks.

Our windowed GUI is not ideal for practical use in Croquet. The icons are too small for being controlled from a certain distance. For a next iteration, we could imagine to design a more simple and intuitive GUI, showing the parameters in a more graphical or even three-dimensional way (eventually inspired by real world interfaces as shown on site [5]).

4.6. Sound File Location

As usual in Croquet, all users have an image with the same code. In our actual version, this also applies to sound files. Every user needs to have exactly the same sound files located in a specified directory. If this is not the case, the generated music will sound different on each computer.

Sound buffer transfer between Croquet images is very slow at the time. We are having problems with TeaReferenceStreams, as soon as we try to exchange large buffers or many small buffers at the same time. So we have to send the sound buffers bytewise, which takes a lot of time (several minutes for a sound buffer of one second length). However, using *compressed audio* (as used successfully in [14]) and an improved Croquet communication system, there could be a chance to let different users come up with their own sound files and distribute them to their friends. Of course, the application would be much more interesting like this.

5. Future Work

It is difficult to work with technologies that are still being built. We have now explored the actual musical possibilities of Croquet and we are ready for refining the integrated features as well as for implementing new ones.

For example, we thought of the possibility to define relations between other objects in the world and generated music. A moving object could play tune and users could detect this object by following this tune. A special kind of UI could even allow users to change this object's tune within the world.

The software we propose could also be used in educational domains. Children could learn to notice and locate musical events. We could then implement a more simple, symbolic and intuitive interface (not necessarily being contained by a window) and integrate educational musical concepts.

Or, as another application example, we could imagine a virtual band as in [14], with the difference that its sound would be three-dimensional. This means that every instrument would be located at a position within the Croquet space. Listeners could move around between these instruments as if they were on stage at a virtual concert. However, this idea cannot be realized using the current implementations of Croquet and OpenAL.

6. Conclusion

In this paper, we have presented our ideas and experiences creating a next generation music software. The current version can be downloaded from: [7]. There may be no professional use for such a software, but in our time, open source online collaboration becomes more and more reputed among home users around the world.

Acknowledgements. We gratefully acknowledge the financial support of the Swiss National Science Foundation for the projects "A Unified Approach to Composition and Extensibility" (SNF Project No. 200020-105091/1, Oct. 2004 - Sept. 2006)

References

[1] Digital musician: Making music online. http://-digitalmusician.net/.

[2] C. Dobrian. Digital audio. http://music.arts.uci.edu/-dobrian/digitalaudio.htm.

[3] ejamming, connect. create. collaborate. jam. no matter where you are. http://www.ejamming.com/.

[4] M. Gaelli. Musinum. http://www.squeaksource.com/.

[5] Tangible user interfaces. http://www.iua.upf.es/mtg/-reacTable/?related.

[6] G. Hiebert. *Creative OpenAL Programmer's Reference*, 2001.

[7] Jam tomorrow: A collaborative composition software for croquet. http://kilana.unibe.ch:8888/JamTomorrow/.

[8] L. Kindermann. Musinum. http://reglos.de/musinum/.

[9] Logic pro. http://www.apple.com/logicpro/.

[10] J. Lombardi. Random acts. http://jlombardi.blogspot.com/-2004/12/random-acts.html.

[11] Model-view-controller. http://en.wikipedia.org/wiki/Model-view-controller/.

[12] My virtual band, open source music and online collaboration. http://www.myvirtualband.com/.

[13] Craig Latta: A musical collaboration network for the internet. http://www.netjam.org.

[14] Ninjam, realtime music collaboration software. http://www.ninjam.com/.

[15] O. W. Sergi Jord. A system for collaborative music composition over the web. In *Proceedings of the 2001 International Computer Music Conference*, La Habana, 2001. International Computer Music Association.

[16] D. A. Smith, A. Kay, A. Raab, and D. P. Reed. Croquet, A Collaboration System Architecture, 2003. in: Proceedings of the First Conference on Creating, Connecting and Collaborating through Computing.

[17] Timidity: Midi to wave converter. http://timidity-.sourceforge.net/.